I just got our consultant’s report. He’s identified our biggest problem.

I recommend that we build a tracking database.

Would you like to hear what the problem is first?

We can put it on the network.

We like databases.

I hate to dwell on the negative.
Lecture Overview

1. SQL introduction & schema definitions
2. Basic single-table queries
3. Multi-table queries
1. SQL INTRODUCTION & DEFINITIONS
What you will learn about in this section

1. What is SQL?

2. Basic schema definitions

3. Keys & constraints intro

4. Activities: CREATE TABLE statements
Basic SQL
SQL Introduction

- SQL is a standard language for querying and manipulating data.

- SQL is a **high-level, declarative** programming language.

- SQL execution is highly optimized and parallelized.

- Many standards out there:
  - Standardized in 1986/87
  - ANSI SQL/ SQL-86, SQL92 (a.k.a. SQL2), SQL99 (a.k.a. SQL3), SQL:2011
  - Vendors support various subsets (e.g., SQLite implements most of the SQL-92 standard)

**SQL** stands for **Structured Query Language**
SQL is a...

- **Data Definition Language (DDL)**
  - Define relational *schemata*
  - Create/alter/delete tables and their attributes

- **Data Manipulation Language (DML)**
  - Insert/delete/modify tuples in tables
  - Query one or more tables
A \textit{relation} or \textit{table} is a multiset of tuples having the attributes specified by the schema.

This is where the name “relational” databases comes from.
### Tables in SQL

- **PName**: Gizmo
  - **Price**: $19.99
  - **Manufacturer**: GizmoWorks
- **PName**: Powergizmo
  - **Price**: $29.99
  - **Manufacturer**: GizmoWorks
- **PName**: SingleTouch
  - **Price**: $149.99
  - **Manufacturer**: Canon
- **PName**: MultiTouch
  - **Price**: $203.99
  - **Manufacturer**: Hitachi

---

**Product**

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

A **multiset** is an unordered list (or: a set with multiple duplicate instances allowed)

- List: [1, 1, 2, 3]
- Set: {1, 2, 3}
- Multiset: {1, 1, 2, 3}

i.e. no `next()`, etc. methods!
## Tables in SQL

An **attribute** (or **column**) is a typed data entry present in each tuple in the relation.

Attributes must have an **atomic** type in standard SQL, i.e. not a list, set, etc.

### Product

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>
# Tables in SQL

## Product

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

A **tuple** or **row** is a single entry in the table having the attributes specified by the schema.

*Sometimes also referred to as a *record*.*
### Tables in SQL

#### Product

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

The number of tuples is the **cardinality** of the relation.

The number of attributes is the **arity** of the relation.
Data Types in SQL

• Atomic types:
  – Characters: CHAR(20), VARCHAR(50)
  – Numbers: INT, BIGINT, SMALLINT, FLOAT
  – Others: MONEY, DATETIME, ...

• Every attribute must have an atomic type

SQLite uses: integer, text and real
Table Schemas

• The **schema** of a table is the table name, its attributes, and their types:

Product(Pname: *string*, Price: *float*, Category: *string*, Manufacturer: *string*)

• A **key** is an attribute (combination) that identifies a tuple uniquely.

Product(Pname: *string*, Price: *float*, Category: *string*, Manufacturer: *string*)
Key constraints

A **key** is a **minimal subset of attributes** that acts as a unique identifier for tuples in a relation.

A key is an implicit constraint on which tuples can be in the relation, i.e., if two tuples agree on the values of the key, then they must be the same tuple!

1. Which would you select as a key?
2. Is a key always guaranteed to exist?
3. Can we have more than one key?

**key candidates and primary key**

Students(sid:string, name:string, gpa: float)
NULL and NOT NULL

- To say “don’t know the value” we use NULL

Students(sid:string, name:string, gpa: float)

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Bob</td>
<td>3.9</td>
</tr>
<tr>
<td>143</td>
<td>Jim</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Say, Jim just enrolled in his first class.

In SQL, we may constrain a column to be NOT NULL, e.g., “name” in this table
General Constraints

• We could specify arbitrary assertions
  – E.g., “There cannot be 25 people in the DB class”

• In practice, we don’t specify many constraints in the database. Why?
  – Performance!
Summary of Schema Information

• Schema and Constraints are how databases understand the semantics (meaning) of data

• They are also useful for optimization

• SQL supports general constraints:
  – Keys and foreign keys are most important
  – We will learn about other constraints
Activities

• SQLite data types
  (http://www.tutorialspoint.com/sqlite/)

• DB Browser
  – Create a database
  – Create a “Product” table
  – Add the shown data

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>
2. SINGLE-TABLE QUERIES
What you will learn about in this section

1. The SFW query

2. Other useful operators: LIKE, DISTINCT, ORDER BY

3. Activities: Single-table queries
SQL Query

• Basic form (there are many many more bells and whistles)

```sql
SELECT <attributes>
FROM <one or more relations>
WHERE <conditions>
```

Call this a **SFW** query.
Simple SQL Query: Selection

**Selection** is the operation of filtering a relation’s tuples on some condition.

```
SELECT * 
FROM   Product 
WHERE  Category = 'Gadgets'
```
**Projection** is the operation of producing an output table with tuples that have a subset of their prior attributes.

```sql
SELECT Pname, Price, Manufacturer
FROM   Product
WHERE  Category = 'Gadgets'
```
Notation

Input schema

Product(PName, Price, Category, Manufacturer)

Output schema

SELECT Pname, Price, Manufacturer
FROM  Product
WHERE  Category = 'Gadgets'

Answer(PName, Price, Manufacturer)
A Few Details

• **SQL commands** are case insensitive:
  – Same: SELECT, Select, select
  – Same: Product, product

• **Values** are not:
  – Different: ‘Seattle’, ‘seattle’

• Use single quotes for text constants:
  – ‘abc’ - yes
  – “abc” - no
LIKE: Simple String Pattern Matching

SELECT * 
FROM   Products 
WHERE  PName LIKE ‘%gizmo%’

•  s LIKE p: pattern matching on strings
•  p may contain two special symbols:
  –  %  = any sequence of characters
  –  _  = any single character
DISTINCT: Eliminating Duplicates

SELECT DISTINCT Category
FROM Product

Versus

SELECT Category
FROM Product

<table>
<thead>
<tr>
<th>Category</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Gadgets</td>
<td></td>
</tr>
<tr>
<td>Photography</td>
<td></td>
</tr>
<tr>
<td>Household</td>
<td></td>
</tr>
</tbody>
</table>
ORDER BY: Sorting the Results

SELECT PName, Price, Manufacturer
FROM Product
WHERE Category='gizmo' AND Price > 50
ORDER BY Price, PName

Ties are broken by the second attribute on the ORDER BY list, etc.
Ordering is ascending, unless you specify the DESC keyword.
Text is ordered alphabetically.
CASE Statement

CASE WHEN [condition1] THEN [expression1]
WHEN [condition2] THEN [expression2]
ELSE [default expression] END

Example:

```
SELECT name,
    CASE WHEN price > 200 THEN 'Yes' ELSE 'No' END AS expensive
FROM Product
```
IN and BETWEEN

The IN operator allows you to specify multiple values in a WHERE clause.

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...)
```

The BETWEEN operator selects values within a range. The values can be numbers, text, or dates.

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2
```
LIMIT Clause

Used to limit the data amount returned by the SELECT statement.

Example: Find the 5 most expensive products

```
SELECT * FROM product
ORDER BY price DESC
LIMIT 5
```

Syntax: LIMIT [no of rows] OFFSET [row num]

Note: LIMIT is not standard SQL (e.g., MS SQL Server uses SELECT TOP)
COUNT

COUNT is an aggregation function that returns the number of elements.

Example: Find the number of products with a price of $20 or more.

```
SELECT COUNT(*) FROM product
WHERE price >= 20
```

Syntax: COUNT([ALL | DISTINCT] expression)
Activities

- SQLite Operators
- Expressions
- Where clauses
- And & Or clauses

(http://www.tutorialspoint.com/sqlite/)

1. How many gadgets are less than $20?
2. How much does it cost to buy all Gadgets?
3. What happens if the manufacturer GizmoWorks changes its name? → This is why we need multiple tables!
3. MULTI-TABLE QUERIES
What you will learn about in this section

1. Foreign key constraints

2. Joins: basics

3. Joins: SQL semantics

4. Activities: Multi-table queries
Foreign Key Constraints

• Suppose we have the following schema:

Students(sid: string, name: string, gpa: float)
Enrolled(student_id: string, cid: string, grade: string)

• And we want to impose the following constraint:
  – ‘Only existing students may enroll in courses’ i.e. a student must appear in the Students table to enroll in a class

<table>
<thead>
<tr>
<th>Students</th>
<th>Enrolled</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>name</td>
</tr>
<tr>
<td>101</td>
<td>Bob</td>
</tr>
<tr>
<td>123</td>
<td>Mary</td>
</tr>
</tbody>
</table>

We say that student_id is a foreign key that refers to Students

Note: student_id alone is not a key- what is?
Declaring Foreign Keys

Students(sid: string, name: string, gpa: float)
Enrolled(student_id: string, cid: string, grade: string)

CREATE TABLE Enrolled(
    student_id CHAR(20),
    cid CHAR(20),
    grade CHAR(10),
    PRIMARY KEY (student_id, cid),
    FOREIGN KEY (student_id) REFERENCES Students
)
Foreign Keys and Update Operations

- What if we insert a tuple into Enrolled, but no corresponding student?
  - INSERT is rejected (foreign keys are constraints)!

- What if we delete a student?
  1. Disallow the delete
  2. Remove all of the courses for that student
  3. *SQL allows a third via NULL (not yet covered)*

---

**Students**

```plaintext
Students(sid: string, name: string, gpa: float)
```

**Enrolled**

```plaintext
Enrolled(student_id: string, cid: string, grade: string)
```

---

**SQLite**: Enable foreign keys with PRAGMA foreign_keys = ON;

**DB Browser**: check “Foreign Keys” in “Edit Pragma”
Keys and Foreign Keys

Company

<table>
<thead>
<tr>
<th>CName</th>
<th>StockPrice</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>25</td>
<td>USA</td>
</tr>
<tr>
<td>Canon</td>
<td>65</td>
<td>Japan</td>
</tr>
<tr>
<td>Hitachi</td>
<td>15</td>
<td>Japan</td>
</tr>
</tbody>
</table>

Product

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

What is a foreign key vs. a key here?
Keys and Foreign Keys

- This example uses **natural keys**.
- Often **surrogate keys** are used instead:

  ```
  Company(CName, StockPrice, Country)
  Product(PName, Price, Category, Manufacturer)
  ```

- Why?
- Why do we use SMUIDs and Social Security Numbers?
Joins

**Product** (PName, Price, Category, Manufacturer)
**Company** (CName, StockPrice, Country)

Ex: Find all products under $200 manufactured in Japan; return their names and prices.

```sql
SELECT PName, Price
FROM Product, Company
WHERE Manufacturer = CName
    AND Country='Japan'
    AND Price <= 200
```

Note: we will often omit attribute types in schema definitions for brevity, but assume attributes are always types.
Joins

Ex: Find all products under $200 manufactured in Japan; return their names and prices.

SELECT PName, Price
FROM Product, Company
WHERE Manufacturer = CName
AND Country='Japan'
AND Price <= 200

A join between tables returns all unique combinations of their tuples which meet some specified join condition.
Several equivalent ways to write a basic join in SQL:

```
SELECT PName, Price
FROM    Product, Company
WHERE   Manufacturer = CName
        AND Country='Japan'
        AND Price <= 200
```

```
SELECT PName, Price
FROM    Product
JOIN    Company ON Manufacturer = Cname
WHERE   Price <= 200
        AND Country='Japan'
```
SELECT PName, Price
FROM  Product
JOIN  Company ON Manufacturer = Cname
WHERE  Price <= 200
       AND Country='Japan'
Tuple Variable Ambiguity in Multi-Table

Person(name, address, worksfor)
Company(name, address)

SELECT DISTINCT name, address
FROM        Person, Company
WHERE       worksfor = name

Which “address” does this refer to?
Which “name”s??
Tuple Variable Ambiguity in Multi-Table

Both equivalent ways to resolve variable ambiguity

Person(name, address, worksfor)
Company(name, address)

SELECT DISTINCT Person.name, Person.address
FROM Person, Company
WHERE Person.worksfor = Company.name

SELECT DISTINCT p.name, p.address
FROM Person p, Company c
WHERE p.worksfor = c.name
An Example of SQL Semantics

```
SELECT R.A
FROM   R, S
WHERE  R.A = S.B
```

R

<table>
<thead>
<tr>
<th>A</th>
<th>1</th>
<th>3</th>
</tr>
</thead>
</table>

S

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

Result

<table>
<thead>
<tr>
<th>A</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
An Example of SQL Semantics

Lecture 2 > Section 3 > Joins: semantics

SELECT R.A
FROM R, S
WHERE R.A = S.B

Cross Product

Apply Selections / Conditions

Apply Projection
Note the **Semantics** of a Join

1. **Take cross product:**
   \[ X = R \times S \]

2. **Apply selections / conditions:**
   \[ Y = \{(r, s) \in X \mid r.A == r.B\} \]

3. **Apply projections** to get final output:
   \[ Z = (y.A,) \text{ for } y \in Y \]

**SELECT** R.A
**FROM** R, S
**WHERE** R.A = S.B

Recall: Cross product (A \times B) is the set of the combinations of all unique tuples in A and B

Ex: \{a,b,c\} \times \{1,2\}
   = \{(a,1), (a,2), (b,1), (b,2), (c,1), (c,2)\}

= Filtering!

= Returning only some attributes

Remembering this order is critical to understanding the output of complicated queries!
Another Note on Semantics

• “semantics” is not equal to “execution order”

• The preceding slides show what a join means

• Not actually how the DBMS executes it under the covers
A Subtlety About Joins

Product(PName, Price, Category, Manufacturer)
Company(CName, StockPrice, Country)

Find all countries that manufacture some product in the ‘Gadgets’ category.

SELECT Country
FROM   Product, Company
WHERE  Manufacturer=CName AND Category=‘Gadgets’
A Subtlety About Joins

Find all countries that manufacture some product in the ‘Gadgets’ category.

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manuf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19</td>
<td>Gadgets</td>
<td>GWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29</td>
<td>Gadgets</td>
<td>GWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

**SELECT Country**
**FROM** Product, Company
**WHERE** Manufacturer=Cname
AND Category=‘Gadgets’

What is the problem? What is the solution?
Activities

1. Create the product/company database from the slide set. Add the following relation

   \[
   \text{Purchase}(\text{id}, \text{product}, \text{buyer}).
   \]

   with the appropriate foreign key constraints and add some data.

2. Find all countries that manufacture some product in the ‘Gadgets’ category (shows each country only once).

3. Find all products that are manufactured in the US sorted by price.

4. In how many different countries are the products a specific buyer purchases manufactured?